*Divizibilitate*

# Divizorii lui n

* Se citește un număr n. Să se afișeze toți divizorii lui n.

Spunem că d este divizor al lui n dacă n se împarte exact la d.

* **Rezolvare:**

Divizorii lui n

#include <

iostream

>

using namespace std;

int main() {

int n, d;

cin

>>

n;

cout<<"Divizorii su

nt:"

;

d = 1;

while ( d <= n ) {

if ( n

% d == 0 )

cout

<<

d

<<

’ ’

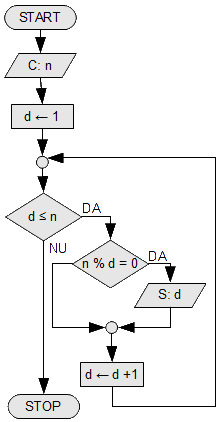
;

d = d + 1;

}

return 0;

}



|  |  |  |
| --- | --- | --- |
| *n* | *d* | *Explicaţii:* |
| *6* | *1* | *iniţializare* |
|  | *2* | *divizor* |
|  | *3* | *divizor* |
|  | *4* |  |
|  | *5* |  |
|  | *6* | *divizor* |
| *6* | *7* | *d>n* |

*Aplicaţii:*

* Se citește un număr n. Scrieţi un program care să afişeze cel mai mare divizor al lui n, diferit de n.
* Se citește un număr n. Scrieţi un program care să afişeze numărul divizorilor lui n.
* Se citește un număr n. Scrieţi un program care să afişeze suma divizorilor lui n.

# Numere prime

 Se citește un număr n. Să se spună dacă n este prim. Un număr este prim dacă nu se împarte decît la 1 și la el însuși.

Număr prim

#include <

iostream

>

using namespace std;

int main() {

int n, d;

cin

>>

n;

d = 2;

while ( d <

n && n

% d > 0 )

d = d + 1;

if ( d == n )

cout<<

"

prim

\

n"

;

else

cout<<

"

ne

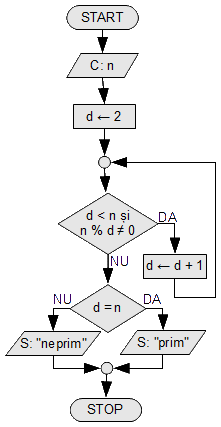
prim

\

n";

return 0;

}



|  |
| --- |
| #include <iostream> using namespace std; int main() { int n, d;  cin>>n; d = 2; while ( d\*d < n && n % d > 0 ) d = d + 1; if ( n%d > 0 ) cout<< "prim\n"; else cout<<"neprim\n";  return 0;  } |

# Numere prime mai mici ca n

* Se citește un număr n. Să se afișeze toate numerele prime mai mici sau egale cu n.
* Rezolvare:

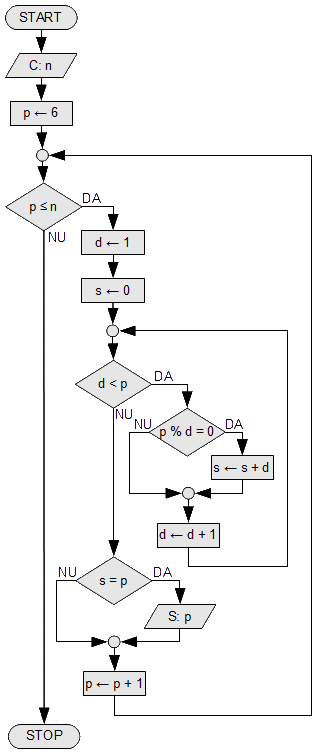
|  |  |
| --- | --- |
|  | #include <iostream> using namspace std; int main() { int n, p, d;  cin<<n; cout<<"Nr. Prime pina la: "<<n; p = 2; while ( p <= n ) { d = 2; while ( d < p && p % d > 0 ) d = d + 1; if ( d == p ) cout<<p<<’ ’; p = p + 1;  } cout<<"\n";  return 0; |
| } |
|  |
| Numere prime până la n |
|  |

*Aplicaţii:*

* Se citește un număr n. Scrieţi un program care să afişeze cel mai mare număr prim mai mic decât n.  Se citește un număr n. Scrieţi un program care să afişeze cel mai mic număr prim mai mare decât n.

# Numere perfecte

* Se citește un număr n. Să se afișeze toate numerele perfecte mai mici sau egale cu n. Un număr este *perfect* dacă este egal cu suma divizorilor lui strict mai mici ca el. Primul număr perfect este 6, deoarece 6 = 1 + 2 + 3. Următorul număr este 28 deoarece 28 = 1 + 2 + 4 + 7 + 14. Exemple: dacă n = 8 se va afișa 6. Dacă n = 30 se va afișa 6 28. Dacă n = 500 se va afișa 6 28 496.
* **Răspuns**:

#include <iostream> using namespace std; int main() { int n, p, d, s;

cin>>n; p = 6; while ( p <= n ) { d = 1; s = 0; while ( d < p ) { if ( p % d == 0 ) s = s + d; d = d + 1;

} if ( s == p ) cout<< p; p = p + 1;

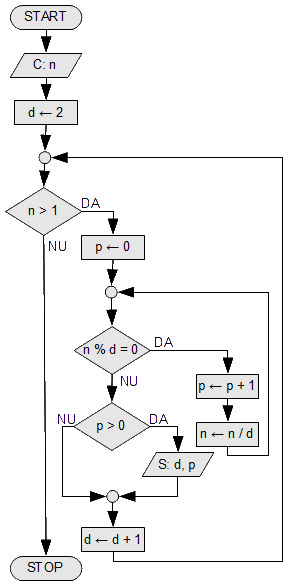
} return 0;

Numere perfecte

# Descompunere în factori primi

Se citește un număr n. Să se descompună în factori primi. Exemple: dacă n = 12 vom afișa 12 = 2^2 \* 3; dacă n = 504 vom afișa 504 = 2^3 \* 3^2 \* 7.

***Notă de rezolvare***: nu este nevoie să testăm dacă un divizor este prim, putem testa toți divizorii în ordine. Numai cei primi vor ajunge să dividă n.

#include <iostream> using namespace std; int main() { int n, p, d;

cin>>n; cout<<n<<”=”; d = 2; while ( n > 1 ) { p = 0;

while ( n % d == 0 ) { p = p + 1; n = n / d;

} if ( p > 0 )

cout<<" \* "<< d<<’^’<<p; d = d + 1;

}

cout<<’\n’;

return 0; }

Descompunere în factori primi

# Algoritmul lui Euclid

[Algoritmul lui Euclid](http://ro.wikipedia.org/wiki/Algoritmul_lui_Euclid) este o metodă eficientă de calcul al celui mai mare divizor comun (CMMDC). El este denumit după matematicianul grec Euclid, care l-a inventat. Un avantaj important al algoritmului lui Euclid este că el poate găsi CMMDC eficient fără să trebuiască să calculeze factorii primi ai numerelor.

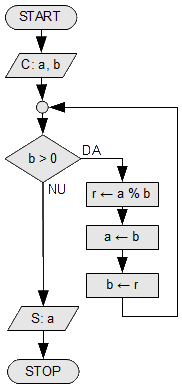
Euclid a observat că CMMDC al două numere a și b rămîne același dacă se scade numărul mai mic din cel mai mare (demonstrați!). Presupunînd că a este numărul mai mare, scăzînd în mod repetat pe b din a, în a va rămîne restul împărțirii lui a la b. Rezultă că CMMDC(a, b) este totuna cu CMMDC(b, a % b). Algoritmul care rezultă este următorul: luăm restul împărțirii lui a la b, apoi restul împărțirii lui b la rest, și așa mai departe, pînă ce obținem un rest zero. CMMDC este numărul rămas, cel diferit de zero.

## Exemplu

Să calculăm CMMDC(252, 105). Calculăm 252 % 105 = 42. De aceea va trebui să calculăm

CMMDC(105, 42). În continuare calculăm 105 % 42 = 21. Deci, vom calcula CMMDC(42, 21). Calculăm 42 % 21 = 0. Deoarece restul este zero CMMDC va fi ultimul număr nenul, adica 21. **Schema logică și programul**

|  |
| --- |
| Notă: remarcați că **nu este nevoie** să interschimbăm variabilele dacă a este inițial mai mic ca b. După prima execuție a buclei while ele se vor interschimba în mod natural. |

#include <iostream> using namespace std; int main() { int a, b, r;

cin>>a>>b; while ( b > 0 ) { r = a % b; a = b; b = r; } cout<<a;

return 0; }

CMMDC cu algoritmul lui Euclid

### *Exercițiu*

Doi prieteni, un iepure și o broscuță joacă un joc: pornesc de la o linie de start și încep să sară. Broasca sare n centimetri, iar iepurele m centimetri. Cine este mai în spate vine la rînd să sară. Jocul se termină atunci cînd iepurele și broasca sînt iarăși la egalitate. Cîți centimetri au parcurs cei doi prieteni?

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | **Răspuns**: după cum se vede și din figură, broscuța și iepurele vor sări o lungime egală cu CMMMC(m, n). Precum știm  CMMMC(m, n) = m \* n / CMMDC(m, n) | |

Săriturile broscuței și iepurelui